CoordiQ : Coordinated Q-learning for Electric Vehicle Charging Recommendation

Carter W. Blum

December 14, 2019

1 Introduction

As the public becomes more ecologically conscious, electric vehicles are surging in popularity, putting up double digit growth rates year over year. However, despite consumer enthusiasm, electric vehicles still face some major hurdles that they must face before they can truly become mainstream. One such hurdle is public charging stations. Because not all electric vehicle owners have the infrastructure to charge their vehicles at home, many rely on electric vehicle charging stations to keep their tank full. This problem is particularly prevalent among electric car owners in urban environments, who often do not have a space to charge their own vehicle.

Unfortunately, electric vehicle charging stations have not spread as prolifically as the vehicles themselves, causing a potential lack of supply. Each charging station has a number of chargers, each of which can service one car at a time. Our data analysis shows that, in the city of Beijing, at least 20% of charging stations are at 100% occupancy - even during periods of low-usage. Electric vehicles can take hours to charge, so stations that are full can stay full for long periods of time, resulting in very long wait times for any vehicles waiting for a charger.

In our increasingly digital world, consumers increasingly rely on mobile apps to direct them to the best utilities. Perhaps the most prominent such utility in China is Baidu Maps, which services 300 million users per month and processes hundreds of thousands of electric vehicle charging station queries per year. Consumers query the app, send it their location, and it sends them back recommendations for nearby charging stations. Current applications simply return a sorted list of the nearest highly rated stations, but this can cause problems, as these are often full, and wait time can cause more of an inconvenience to the user than driving time.

This project implements and evaluates a reinforcement learning agent to intelligently respond to queries, recommending users to nearby charging stations with minimal inconvenience. The project assumes no control over when the app will receive queries and it assumes that there are many other electric vehicles that it has no control over. Despite these constraints, we show that the agent is able to learn effective recommendations that reduce the user's time-to-charging by over 45%.

The key contributions of this work are as follows :

- We present the first machine learning based algorithm for charging station recommendation, and implement a context-aware reinforcement algorithm that coordinates between different recommendations. It substantially outperforms all existing baselines.
- To the best of our knowledge, there are no published works yet using Graph Neural Networks (GNNs) to solve reinforcement learning tasks. We propose an architecture and show it improves performance

2 Related Works

Reinforcement learning is a useful paradigm for solving sequential control tasks of many times. Reinforcement learning models operate in a Markov Decision Process, learning a probability distribution over a finite set of actions to maximize reward [Russell and Norvig, 2016][Sutton and Barto, 2018]. With the recent increase of processing power and the modeling power of neural networks, Deep Q-Networks (DQNs) have shown impressive capabilities to handle a diverse range of tasks [Mnih *et al.*, 2013]. To that extent, Deep Q-Networks have been shown to learn complex tasks, and have even been able to excede human level control when solving many problems [Mnih *et al.*, 2015].

Reinforcement Learning has been successfully applied to improving charging of electric vehicles. However, most of the research has focused on improving the charging experience once a vehicle has arrived in a station, such as in [Valogianni *et al.*, 2013], [Xiong *et al.*, 2018] and [Zou *et al.*, 2016]. These methods successfully improve load distribution equity in stations, and schedule charging demands accordingly, as in [Zhao *et al.*, 2018]. [Ramachandran *et al.*, 2018] also analyze charging station demand to improve the layout of stations to better handle peak demand.

However, reinforcement learning has been applied to electric vehicle station recommendation as well. [Houbbadi *et al.*, 2019] provides a thorough analysis of bus charging recommendations. However, the methods provided focus specifically on buses, following strict and predictable schedules, and the primary difficulty is in choosing when to charge, not where. The question being approached in this paper is the opposite, we assume that the user needs to charge now and the recommender attempts to minimize the inconvenience to the user. On the other hand, [Tian *et al.*, 2016] focuses on choosing what stations to recommend taxis to. The authors employ a rule-based method to do recommendation, but the method provided relies on future knowledge of the occupancy of a charging station at the time that the user arrives. Furthermore, it assigns each taxi in isolation, without accounting for the impact that one taxi's recommendation may have on the wait-time of another.

Outside of electric vehicle recommendation, there is a large body of work in spatio-temporal reinforcement learning. In [Xu *et al.*, 2018], the authors use a tabular method to recommend idle taxis to nearby regions and allocate dispatches. While this approach was extremely successful for its authors,

it heavily exploits the interchangeability of taxis, a property that electric vehicle charging stations do not have to the same degree.

A non-tabular approach has also been pursue in several works, including [He and Shin, 2019], which implements a capsule network to recommend idle taxi cruising. This approach is also modified and expanded in several similar papers, including [Jindal *et al.*, 2018], [Oda and Joe-Wong, 2018], [Oda and Tachibana, 2018], [Shou *et al.*, 2019], [Alabbasi *et al.*, 2019]. These papers improve taxi idle cruising to reposition them to nearby areas, either by considering the taxis in concert or in isolation. They train reinforcement learning models in simulations, with space discretized in to tiles, and the use a convolutional neural network architecture to output Q-values for each location, and then selecting the highest Q-value from the adjacent tiles. However, most tiles are not valid actions for station recommendation, giving this method a very high noise to signal ratio when applied to charging station recommendation.

A couple other approaches to spatio-temporal recommendation have also been used. One approach is to try to minimize the divergence between the distribution of queries and taxi locations, as demonstrated in [Zhou *et al.*, 2019]. While this approach appears very promising, the number of 0 support regions in both query and station distributions causes many distributional distance metrics to be unstable. Another direction is explored in [Verma *et al.*, 2017], which uses a tabular method but dynamically splits the zones whenever it can improve Q-value accuracy. This method can be suitable for macro-planning, but charging station recommendation requires distinguishing between stations that are close together. Under this paradigm, those stations may be grouped in the same zone, resulting in decreased performance.

An alternate method for handling spatially related data is to use graphical neural networks. Graphical neural networks define the input as a graph, with vertices edges and weights. This allows information from nodes to propagate from one to another [Scarselli *et al.*, 2008]. Initial methods put restrictions on the weights of the graph convolution kernels to ensure that repeated convolution would converge [Pineda, 1987][Defferrard *et al.*, 2016]. However, more recently methods methods have been doing away with these restrictions [Kipf and Welling, 2016]. One such method that has had significant success is GraphSage, which utilizes local graph convolution over a finite set of iterations. This method does not guaranteed converge to a set value, but can, in practice achieve similar, if not superior results with significantly less computation.

3 Methodology

We formulate the problem as a Markov Decision Process (MDP), with a statespace $\mathcal{S}$, action space $\mathcal{A}$, transition function T, and reward function R. Our recommendation system is represented by a policy $\pi : \mathcal{S} \rightarrow \mathcal{A}$ At each timestep in an MDP, an agent must look at it's current state and choose an action to maximize its expected discounted rewards. The expected rewards of action $a \in \mathcal{A}$ while at state $s \in \mathcal{S}$ is given by :

$$Q(s,a) = E[R(t)] + \gamma \sum_{s' \in \mathcal{S}} T(s'|s,a) \max_{a' \in \mathcal{A}} Q(s',a')$$

here $\gamma \in [0,1)$ is the discount factor.

States: In this problem, the state is the information available to the station to make recommendations with. Some information is available for each city that may be relevant to all stations, such as estimates of traffic, the weather, day of week, and time of day. For a given state s, we'll refer to this global information as s_G. For each of the N stations, information about the status of its chargers is available, as well as its location and id. The information available about the n^{th} station is denoted as s_n. Finally, the agent has access to information about the queries it is responding to, including their location and user id, as well as time of query. For the i^{th} query received at time t, the relevant state information is denoted as s_i.

We discretize the each regionin to a grid, with each grid cell having an area of roughly 0.25km^2. We similarly turn the system in to a sequential decision process by dividing time in to non-overlapping intervals of duration τ.

Actions: At each timestep t, the agent receives a set of user queries Q_t for which it must provide recommendation. The station can recommend each of the $|Q_t|$ user queries to any of the N stations, resulting in a possible action space of $N^{|Q_t|}$ combinations. This poses a problem, because not only is the possible action space potentially huge, but the available actions vary from timestep to timestep. To address this, the agent makes each of its decisions for each query sequentially and executes all actions after decisions about all queries have been made. When there is no loss of clarity, we will instead refer to recommending a user to station n as action a_n.

Rewards: After a driver is recommended to a station, they must drive there. Subsequently, if the station is full, then they must spend time waiting for a space to open up. To accommodate these requirements, the agent must then satisfy the multi-modal objective of minimizing both of these. Let t^i_{drive} be the driving time for the user corresponding to query i, as estimated by driving speed and distance. Similarly, we define t^i_{wait} to be the the number of timesteps that the user spends waiting at the station before a spot opens for them. As the agent wants to find an optimal split between users, the goal is to then minimize the following equation

$$\sum_t \sum_{i \in \mathcal{Q}(t)} t^i_{wait} + \lambda t^i_{drive}$$

where λ is a factor for weighting the two criteria. We define the sum of these two items (with $\lambda = 1$) to be the inconvenience time for a user. To achieve this, the reward function is defined as

$$R(t) = KN_{arrive} - N_{wait} - \lambda N_{drive}$$

Where N_{arrive}, N_{wait} and N_{drive} are defined as the number of users who arrived at a station, waited there or were driving at the current timestep, respectively. K is the reward that the agent receives for successfully completing a dispatch. Trivially, as $\gamma \rightarrow 1$, maximizing this reward corresponds to minimizing the previous equation.

Transition Function: In this study, the transition function is modeled by a simulation, which is further described in the 'Analysis & Results' section. In practice, the transition function will be the real-world movement of cars to and from stations, as well as the updated information that the agent receives from these actions. Note that this simulation assumes that there are many vehicles that leave and enter the station outside of our control, and these vehicles must be considered part of the transition function. Once a car recommended by our station begins charging, we have no further influence over it and it becomes part of the environment.

However, he above formulation does not quite fulfill the requirements for a Markov decision process, which requires that the transition function is solely a function of the current state, because of several reasons. First, how many cars enter and leave a given charging station may be influenced by how long those cars have already been charging. In the analysis section, we show that, counterintuitively, this is not the case.

Second, how many cars enter a station is partially influenced by our previous recommendations. To account for this, we add an additional engineered feature included in the station information (s_n). For each $\delta t \in [1, k]$, we include information about the number of cars that are estimated to arrive at timestep $t + \delta t$ as a result of our recommendations. As this information can be updated by the agent every timestep *and* whenever it makes a decision, it allows the agent to remember its queued decisions that it hasn't executed yet. As a result of this, it is able to coordinate between the many decisions it can be forced to make at each timestep. Koenig et Al. show that an optimal policy learned this way will obtain no less than half of the rewards of an optimal policy that considers all $\mathcal{Q}(t)$ at once, while avoiding the exponential complexity of such an algorithm.

Classical Model: A classical method of solving such problems is to use a feedforward Deep Q-Network (DQN). In this case, the state is input as a concatenation of all of the parts of the station $s = \{s_n \forall n, s_i \forall i \in \mathcal{Q}(t), s_G\}$. The DQN, parameterized by θ, outputs a vector $\vec{q}$, such that $\vec{q}_n$ denotes the Q-value for action a_n. The policy π, is to then usually select the station with the highest Q-value. However, to incentivize exploration, the agent sometimes chooses a random agent with probability ϵ. The network is then trained via temporal differencing, such that the update is calculated as follows:

$$\Delta\theta = \eta \nabla_\theta \Big(R(t) + \max_{a' \in A} Q(s', a' | \Theta') - Q(s, a | \Theta) \Big)$$

To improve convergence, a replay memory buffer is used to sample old experiences to perform the TD-update on.

Convolutional Model: There can be hundreds of stations, meaning that this single network must take in all of the hundreds on inputs at once and map this to the hundreds of appropriate Q-values. However, he occupancy has of a station on the southern side of the city likely should have little influence on the agent's evaluation of an arbitrary station on the other side of the city. Furthermore, the evaluation process should be roughly the same for each station - look at how far it is, look at whether or not we anticipate a demand in the area, etcetera. As a result, the network can be improved by evaluating each station individually. In this case, rather than concatenating all of the states, when calculating $Q(s, a_n)$ for the i^{th} query, the agent is only input with the values of $\{s_n, s_i, s_G\}$. In this formulation, the evaluation of all of the stations shares the same parameters.This also enables the agent to operate with far fewer parameters, speeding up computation. Mathematically, this operation is equivalent to inputing the state as an $Nx1$ image, with depth equal to the number of features for each station plus the number of features for the global state. Processing each station one at a time then corresponds to doing 1x1 convolution over the stations, hence the name 'convolutional model'.

Algorithm 1 Code for agent actions

ACT(π, S, $\mathcal{Q}(t)$, E)

SHUFFLE($\mathcal{Q}(t)$)
for $q^{(j)} \in \mathcal{Q}(t)$ **do**
  for $s^{(i)} \in$ NEARBY$(q^{(j)}, S)$ **do**
    bid$^{(i)} = Q(s^{(i)}, q^{(j)}, E)$
  end for
  to = argmax$_j \Big[$bid$^{(i)}\Big]$
  dist = DISTANCE($s^{(to)}$, $q^{(j)}$)
  $s^{(i)}_{d_{dist}} += 1$
end for

Graphical Model: Despite these advantages, this analysis can be slightly reductive. Evaluating each station in isolation could potentially miss out on important information - if all other nearby stations are full, the remaining station with vacancies can expect a lot of their demand to be redirected to it. It is then useful for the station to have a method to view information about nearby states.

Given that the spread of stations is non-uniform, traditional convolutional methods do not work. As a result, this paper employs graph convolution to add context [Kipf and Welling, 2016]. Specifically, this paper employs a form of graph convolutional networks similar to GraphSage. Specifically, we define a graph $G = \{V, E, W\}$ over the state space. Specifically, we let the set of vertices V to be the set of stations and add edges between two nodes if the corresponding stations are within distance D from each other. Letting $d(n, m)$ be the distance between stations n and m, the corresponding weight is defined as $\alpha_n m e^{-\beta d(n,m)}$, where α is a hyperparameter.

Denoting $E(n)$ as the stations that share an edge with n, the Q-values of action n are computed as follows.

$$h_n = f(s, a_n | \theta_1)$$

$$x_n = \{h_n, \sum_{m \in E(n)} \alpha_m h_m\}$$

$$q_n = g(x_n | \theta_2)$$

where f, g are feed-forward networks. A diagram explaining each of the three model types is provided in figure (1) and the pseudocode provides an overview of the sequential decision process.

An alternate way to formulate the problem is as a multi-agent problem. In this formulation, each station could be formulated as its own agent, and for each query, it has a decision

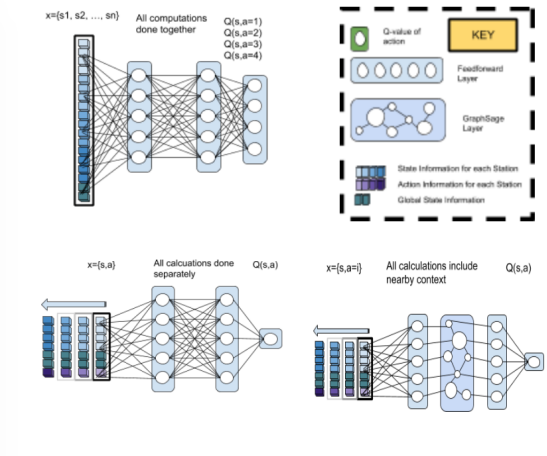

Figure 1: Depictions of easy model type. *Top-Left:* Classical FFDQN network *Bottom-Left:* Convolutional Model *Bottom-Right:* GraphSage Model

to accept or not accepted the user to be recommended to it. In this way, each station receives reward for the queries that are dispatched to it. However, our MDP requires that the agent can only recommend one station at a time, so there needs to be consensus between stations, such that exactly one station 'accepts' each query.

Let $Q_n(s,1)$ be the discounted expected reward of station n if it accepts, and $Q_n(s,0)$ if it rejects. Then maximizing the total reward over all stations is maximized by recommending the user to station n such that

$$n = \text{argmax}_{n \in N} Q_n(s,1) - Q_n(s,0)$$

We explore this model in analysis and results below.

To make any of these models learn anything at all, several small improvements are also necessary. First, the agent can only recommend agents to stations within distance D. If there are less than 5 stations within distance D, the agent can recommend to the 5 nearest stations instead. Additionally, to stabilize training, this paper uses the Double DQN architecture, as pioneered in [**?**] . Finally, a dueling DQN architecture was implemented, using a convolutional neural network to learn the value function.

4 Experiment Design

Data Analysis: The algorithm is trained and tested in a simulation based on data from over 6 months of data. As noted above, the region is divided in to a rectangular grid. In the case of Beijing, the city is divided in to a grid of size 144×126. Time is discretized in to non-overlapping periods, each with duration $\tau = 15$ minutes. Charger data is collected 514 chargers across 73 stations and 96,000 timesteps. This is used to measure how long cars stay at a station by measuring the time between when when a charger becomes occupied and when it first becomes free again. From this, a probability distribution of charging durations is derived, as shown in figure (2). We note that the distribution appears to roughly follow a geometric distribution (p ¡ 5e-4), which has the convenient

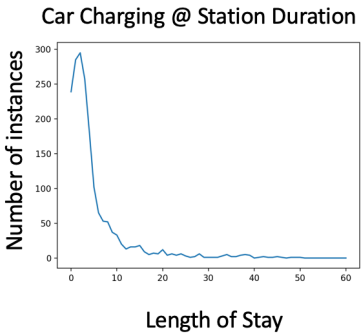

Figure 2: Counts of charging durations by number of observations

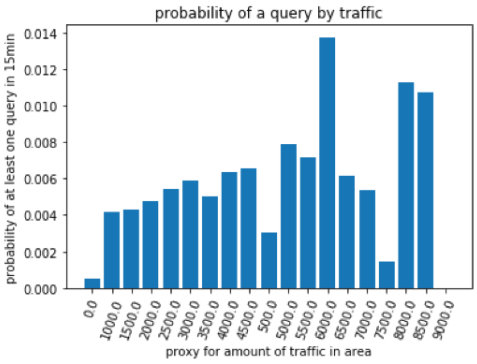

Figure 3: The number of queries received in an area doesn't correlate strongly with the number of cars in the area. Note that there is higher variance for higher-traffic data values, as there are fewer datapoints.

property of being memoryless. This means that knowing how long a vehicle has been charging for does not help predict how much longer it will continue charging, so the model does not need to keep track of this information.

As a result, at each timestep, we can model the number of remaining vehicles as a geometric distribution. The next question is the number of incoming cars. Naturally, one would expect that the number of cars coming in to the station and the number of cars querying would be proportional to the traffic in the surrounding area. However, this assumption does not match the data, as shown in figure (3). As nearby traffic doesn't seem to correlate strongly with any core pieces of the environment, it is also not included in the state.

Simulation: With these considerations, the simulation is defined as follows. The agent is trained in an episodic manner, with each simulation starting and ending at midnight. The number of cars that remain at the station at each timestep is sampled from a geometric distribution. From there, a number of cars incoming from outside for the system is sampled from a distribution modeled off of historical data. Finally, the dispatches made by our system are handled. If possible, all of the vehicles that are within 1 timestep of their target station move to that station and begin charging. If there are not enough chargers, then those vehicles that couldn't be accommodated wait until the next timestep (at which point they will

Algorithm 2 Code for training simulation

SIMULATE(f, g, π, S)

Let $t = 0$.
while $t < T_{END}$ **do**
  for $s^{(i)} \in S$ **do**
    $N^{(i)}_{cars}(t+1) = N^{(i)}_{rem} + N^{(i)}_{inc} + N^{(i)}_{rec}$
    if $s^{(i)}_T < N^{(i)}_{cars}(t+1)$ **then**
      $s^{(i)}_{d_0} += N^{(i)}_{cars}(t+1) - s^{(i)}_T$
      $N^{(i)}_{cars}(t+1) = s^{(i)}_T$
    end if
    $s^{(i)}_O = s^{(i)}_T - N^{(i)}_{cars}(t+1)$
  end for
  $\mathcal{Q}(t) \sim g(t, dow)$
  ACT(π, S, $\mathcal{Q}(t)$)
end while

get priority over arriving vehicles). Finally, the position of all other vehicles is updated. A summary of this algorithm is provided in pseudo code.

5 Results & Analysis

We evaluate the proposed models in the simulation defined above. The convolutional and graphical models are build with a small network size of only 100 nodes per hidden layer and 3 hidden layers. A variety of network sizes were tried for the classical DQN, but the best performing network was substantially larger, with 250 nodes per hidden layer. The exploration probability, ϵ, is initially set at 0.9 and decreases exponentially to 0.1 throughout training.

Aside from the classic DQN model, we evaluate our model against several rule-based metrics, defined as follows:

- **Nearby:** Always recommends the user to the nearest station in a greedy manner. This should minimize t_{drive} for all users.
- **Open:** Always recommends the user to the station that has the highest number of open chargers, breaking ties by distance. This would ideally minimize t_{wait} for all users.
- **NearestOpen:** An approximation of the method described in [Tian *et al.*, 2016], but avoids using future information. Will usually choose nearest station, but will choose other stations if the desired station is full.

We test 4 different deep-learning based methods: Graphical, Convolutional and Classical DQN are all as described above. 'Grouping' denotes a convolutional model that first recommends the user to a group of stations and then uses greedy nearby recommendation once the user arrives. Figure (4) shows a plot of the average rewards received on the test seed by epoch. Note that 'Open' is not shown on graphs because it performs so poorly that it makes the other models hard to compare.

The results shown strongly suggest that classical DQN methods are not able to solve this problem to any satisfying degree, as it performs significantly worse than both rule-based baselines. However, both the convolutional and graphical models substantially outperform the baselines. However, there are a couple of interesting things to note about their performance.

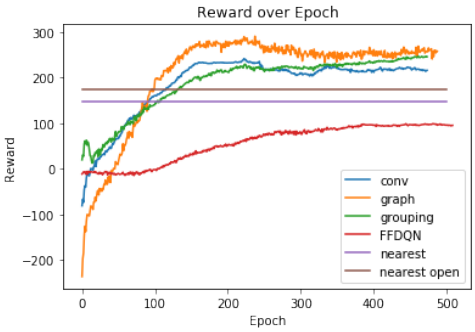

Figure 4: Convolutional and Graphical Models Substantially Outperform Other Strategies

Firstly, there is an interesting phenomenon where the average reward nearly monotonically increases each epoch, but then there is a slight downward dip after roughly epoch 200, before the rewards start climbing upwards again. There are two possible reasons for this. The more interesting reason, is that it may be an instance of a broader phenomenon, called 'Deep Double Descent', where neural networks often undergo a period where their performance decreases after an initial rapid decrease [Mei and Montanari, 2019].

The second possible reason is a bit more banal - it has to do with random seed. The data is tested on random seeds zero through ten. The agent initially plays the simulation with random seed zero, and each time it plays, it increases the random seed by one. Because of this, it does initially see the random seeds that it is eventually tested on, once for each. However, this experiment uses a experience replay buffer to stabilize convergence, and epoch 200 is roughly where those 'memories' would leave the buffer. I doubt that this is having any strong effect, because the policy that the agent followed on its first attempt is likely very different from its final policy, but it is something to be aware of for future experiments.

The other thing to note is that the graphical model does indeed outperform the baseline convolutional model, but it's not the only one to do so. Adding grouping to the convolutional model appears to achieve roughly the same final results, although it doesn't converge as quickly. These experiments take a long time to run (sometimes up to a week), so I would like to test whether or not combining the graphical model and the grouping mechanism results in an even better model. The natural initial response to this question might be a hesitant yes, but, when looking at the wait times, it's not totally clear that it is possible to improve.

For this, we reference figure (**??**). We again note that the theoretical minimum drive distance is equal to the drive distance given by the 'nearest' recommender. Looking at the graph, we see that the graphical and grouping models are actually very close in this regard - the average drive time is only 0.1 and 0.3 timesteps more than that of the 'nearest' recommender. Despite this, the graphical model achieves near-zero wait time. Impressively, the graphing model achieves a re-

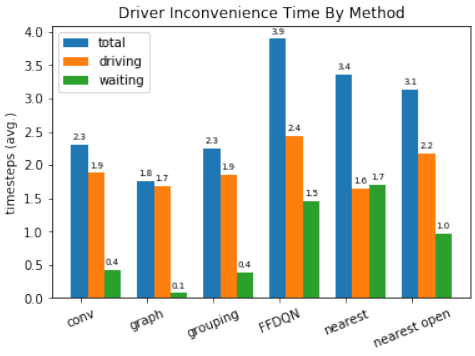

Figure 5: Graphical Model Surprisingly Performs Near-Optimally

sult that is nearly as good as a fantasy world where every user drives to the nearest station and there's always a spot for them - it's average wait time is only 0.2 more than that!

This indicates that, while the graphical model might not be perfect, there likely isn't much to gain from improving it further, as you could only reduce wait times by another 10% at most. To contrast this, the graphical model reduces wait times by about 45% over all of the baselines, which is much more substantial. These results are condensed in table (**??**). However, there are a number of caveats to these statements. The first, and biggest one, is that these are only single runs for each model. Reinforcement learning models are notorious for being unstable, and that has been my experience with this project as well. Its very possible that the random seeds gave good initial weights that helped things converge smoothly. Going forward, I want to run more tests so that I can have proper confidence intervals on these plots.

The second caveat is that these results are within a simulation, and it may be significantly easier to optimize the simulation than the real world. I can't seen any glaring reasons why that would be the case, but it is a common difficulty that many reinforcement learning projects have. That being said, it is very common for reinforcement learning papers to only report in-simulation results, so this is far from out of the norm.

The last set of experiments to touch upon are the multi-agent experiments. Unlike the graphical model, these heavily underperformed. While it did outperform the baseline, it still wasn't anywhere close to the convolutional model and it trained extremely slowly. I can't find a solid justification for this, and would like to investigate it in the future. A plot of the reward functions is provide in figure (6).

I do have a theory as to why this could be happening. The first theory is that the multi-agent model doesn't quite fully satisfy the MDP property. Whenever it does an auction, the model doesn't 'know' if it will have another query this timestep or not, even though that is pre-determined, therefore making it suboptimal. That being said, this same effect should apply to all of the other models as well, but those performed fine. Another possibility is that it simply takes longer to converge, or that the non-stationarity of the other agents makes the environment itself non-stationarity. According to [Foerster *et al.*, 2017], the fact that all agents share parameters should prevent that from happening, but it's possible that something still went wrong.

6 Conclusion and Future Work

In total, this report presents a framework for significantly improving charging station recommendations in response to dynamic, user-generated queries. The graphical model proposed is the first of its kind, and it balances between providing the canvas for complex interactions to be discovered and limiting the search space to allow it to tackle more difficult problems.

I think that there are three primary directions of future work that could be expanded from this. The first is strictly application focused. This model needs to be stress-tested on multiple cities over multiple runs, and could be implemented in a real-world system. To do so, we would likely want to collect more statistics, such as what percentage of users have to wait, and how long the longest wait times are. There are always more experiments to be run.

The second direction is to address the multi-agent problem. In theory, a multi-agent should converge extremely quickly (I did a proof of it in the tabular environment, but it ended up not being relevant, so I didn't add it here). I think that this aspect could very much do to be expanded, and it is highly extendable. For that matter, I think that the graphical method here is highly extensible, most immediately to any point-of-interest recommendation system or other graph-based domain.

The final direction is the most theoretical, but has a lot of potential. In theory, everything done by this network could be down by a fully-connected network, but no network ever came close to achieving these results. As I see it, feature-engineering and simulation aside, all that this paper did is tie weights together and prune weights in the fully-connected network. Frankly, I think that's all that most deep learning papers I read amount to. It would be wonderful if there wer e a system that could detect the pattern in the inputs (that every k dimensions represented a station) and automatically do this pruning and weight-tying. At some level, that's really what abstraction is, and I think automating it could be a big step forward in both model capability but also explainability.

7 Bibliography

References

[Alabbasi *et al.*, 2019] Abubakr Alabbasi, Arnob Ghosh, and Vaneet Aggarwal. Deeppool: Distributed model-free algorithm for ride-sharing using deep reinforcement learning. *arXiv preprint arXiv:1903.03882*, 2019.

[Defferrard *et al.*, 2016] Michaël Defferrard, Xavier Bresson, and Pierre Vandergheynst. Convolutional neural networks on graphs with fast localized spectral filtering. In *Advances in neural information processing systems*, pages 3844–3852, 2016.

[Foerster *et al.*, 2017] Jakob Foerster, Nantas Nardelli, Gregory Farquhar, Triantafyllos Afouras, Philip HS Torr, Pushmeet Kohli, and Shimon Whiteson. Stabilising experience replay for deep multi-agent reinforcement learning. In *Proceedings of the 34th International Conference on*

	Graphical	Convolutional	FFDQN	Nearest	Open	Nearest Open
Reward	**291**	242	97	148	-910	173
Inconvenience Time	**1.76**	2.31	3.90	3.35	15.1	3.08
Wait Time	**0.08**	0.43	1.46	1.70	3.38	0.962
Drive Time	1.69	1.881	2.43	**1.6**	12.65	2.16

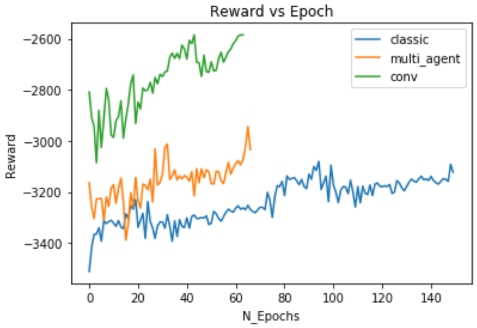

Figure 6: Multi-Agent Model Mysteriously Underperforms

Machine Learning-Volume 70, pages 1146–1155. JMLR. org, 2017.

[He and Shin, 2019] Suining He and Kang G Shin. Spatio-temporal capsule-based reinforcement learning for mobility-on-demand network coordination. In *The World Wide Web Conference*, pages 2806–2813. ACM, 2019.

[Houbbadi *et al.*, 2019] Adnane Houbbadi, Rochdi Trigui, Serge Pelissier, Eduardo Redondo-Iglesias, and Tanguy Bouton. Optimal scheduling to manage an electric bus fleet overnight charging. *Energies*, 12(14):2727, 2019.

[Jindal *et al.*, 2018] Ishan Jindal, Zhiwei Tony Qin, Xuewen Chen, Matthew Nokleby, and Jieping Ye. Optimizing taxi carpool policies via reinforcement learning and spatio-temporal mining. In *2018 IEEE International Conference on Big Data (Big Data)*, pages 1417–1426. IEEE, 2018.

[Kipf and Welling, 2016] Thomas N Kipf and Max Welling. Semi-supervised classification with graph convolutional networks. *arXiv preprint arXiv:1609.02907*, 2016.

[Mei and Montanari, 2019] Song Mei and Andrea Montanari. The generalization error of random features regression: Precise asymptotics and double descent curve. *arXiv preprint arXiv:1908.05355*, 2019.

[Mnih *et al.*, 2013] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Alex Graves, Ioannis Antonoglou, Daan Wierstra, and Martin Riedmiller. Playing atari with deep reinforcement learning. *arXiv preprint arXiv:1312.5602*, 2013.

[Mnih *et al.*, 2015] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A Rusu, Joel Veness, Marc G Bellemare, Alex Graves, Martin Riedmiller, Andreas K Fidjeland, Georg Ostrovski, et al. Human-level control through deep reinforcement learning. *Nature*, 518(7540):529, 2015.

[Oda and Joe-Wong, 2018] Takuma Oda and Carlee Joe-Wong. Movi: A model-free approach to dynamic fleet management. In *IEEE INFOCOM 2018-IEEE Conference on Computer Communications*, pages 2708–2716. IEEE, 2018.

[Oda and Tachibana, 2018] Takuma Oda and Yulia Tachibana. Distributed fleet control with maximum entropy deep reinforcement learning. 2018.

[Pineda, 1987] Fernando J Pineda. Generalization of back-propagation to recurrent neural networks. *Physical review letters*, 59(19):2229, 1987.

[Ramachandran *et al.*, 2018] Anshul Ramachandran, Ashwin Balakrishna, Peter Kundzicz, and Anirudh Neti. Predicting electric vehicle charging station usage: Using machine learning to estimate individual station statistics from physical configurations of charging station networks. *arXiv preprint arXiv:1804.00714*, 2018.

[Russell and Norvig, 2016] Stuart J Russell and Peter Norvig. *Artificial intelligence: a modern approach*. Malaysia; Pearson Education Limited,, 2016.

[Scarselli *et al.*, 2008] Franco Scarselli, Marco Gori, Ah Chung Tsoi, Markus Hagenbuchner, and Gabriele Monfardini. The graph neural network model. *IEEE Transactions on Neural Networks*, 20(1):61–80, 2008.

[Shou *et al.*, 2019] Zhenyu Shou, Xuan Di, Jieping Ye, Hongtu Zhu, and Robert Hampshire. Where to find next passengers on e-hailing platforms?-a markov decision process approach. *arXiv preprint arXiv:1905.09906*, 2019.

[Sutton and Barto, 2018] Richard S Sutton and Andrew G Barto. *Reinforcement learning: An introduction*. MIT press, 2018.

[Tian *et al.*, 2016] Zhiyong Tian, Taeho Jung, Yi Wang, Fan Zhang, Lai Tu, Chengzhong Xu, Chen Tian, and Xiang-Yang Li. Real-time charging station recommendation system for electric-vehicle taxis. *IEEE Transactions on Intelligent Transportation Systems*, 17(11):3098–3109, 2016.

[Valogianni *et al.*, 2013] Konstantina Valogianni, Wolfgang Ketter, and John Collins. Smart charging of electric vehicles using reinforcement learning. In *Workshops at the Twenty-Seventh AAAI Conference on Artificial Intelligence*, 2013.

[Verma *et al.*, 2017] Tanvi Verma, Pradeep Varakantham, Sarit Kraus, and Hoong Chuin Lau. Augmenting decisions of taxi drivers through reinforcement learning for improving revenues. In *Twenty-Seventh International Conference on Automated Planning and Scheduling*, 2017.

[Xiong *et al.*, 2018] Rui Xiong, Jiayi Cao, and Quanqing Yu. Reinforcement learning-based real-time power management for hybrid energy storage system in the plug-in hybrid electric vehicle. *Applied energy*, 211:538–548, 2018.

[Xu *et al.*, 2018] Zhe Xu, Zhixin Li, Qingwen Guan, Dingshui Zhang, Qiang Li, Junxiao Nan, Chunyang Liu, Wei Bian, and Jieping Ye. Large-scale order dispatch in on-demand ride-hailing platforms: A learning and planning approach. In *Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, pages 905–913. ACM, 2018.

[Zhao *et al.*, 2018] Pu Zhao, Yanzhi Wang, Naehyuck Chang, Qi Zhu, and Xue Lin. A deep reinforcement learning framework for optimizing fuel economy of hybrid electric vehicles. In *2018 23rd Asia and South Pacific Design Automation Conference (ASP-DAC)*, pages 196–202. IEEE, 2018.

[Zhou *et al.*, 2019] Ming Zhou, Jiarui Jin, Weinan Zhang, Zhiwei Qin, Yan Jiao, Chenxi Wang, Guobin Wu, Yong Yu, and Jieping Ye. Multi-agent reinforcement learning for order-dispatching via order-vehicle distribution matching. In *Proceedings of the 28th ACM International Conference on Information and Knowledge Management*, pages 2645–2653. ACM, 2019.

[Zou *et al.*, 2016] Yuan Zou, Teng Liu, Dexing Liu, and Fengchun Sun. Reinforcement learning-based real-time energy management for a hybrid tracked vehicle. *Applied energy*, 171:372–382, 2016.

8 Appendix